Defining an Ontological Framework for Modelling Policies in Cloud Environments

Simeon Veloudis and Iraklis Paraskakis
South East European Research Centre (SEERC)
The University of Sheffield, International Faculty CITY College
Thessaloniki, Greece

***Abstract*—Cloud computing enables enterprises to realise significant cost savings whilst accelerating the development of new innovative applications. Nevertheless, due to security concerns, enterprises are reluctant to migrate their operations to the cloud. In addition, the proliferation of cloud services transforms the enterprise IT environment into a complex ecosystem of collaborating services. One way to tame this complexity and alleviate the security concerns is to rely on *policies* that regulate the deployment, delivery and governance of cloud services. However, the heterogeneity inherent in such services, coupled with the dynamic nature of cloud environments, hinders the formulation of effective and interoperable policies. This paper proposes a *generic* framework for the definition and representation of policies that are enforceable across diverse administrative domains and are amenable to automated correctness checks.**

Keywords—Policy; Linked USDL; Security; Cloud Computing

I. Introduction

By embracing the cloud computing paradigm, organisations gain access to a vast ecosystem of infrastructure, platform, and software resources that are abstracted as services and delivered over the Internet on an on-demand basis by diverse providers [1]. The low cost at which these services are typically provisioned enables organisations to realise significant savings whilst accelerating the development and deployment of new applications. Cloud computing therefore acts as a catalyst for innovation and stimulates the introduction of new business models.

Nevertheless, due mainly to confidentiality, privacy and integrity concerns, enterprises are reluctant to migrate their sensitive data to the cloud [2] [3]. In addition, as externally-sourced services proliferate, the enterprise IT environment becomes a complex ecosystem of heterogeneous services, making it increasingly difficult to keep track of when and how services evolve over time, either through intentional changes initiated by their providers, or through unintentional changes, such as fluctuations in service performance and availability [4]. One way to tame this complexity and assist alleviating security concerns, is to rely on policies that regulate the deployment, and governance of cloud services, thereby achieving an adequate level of security and predictability in service behaviour. These policies, if they are to be effective, must: (i) take into account the dynamicity and unpredictability of cloud environments, as well as the inherent heterogeneity of cloud services; (ii) be interoperable – hence enforceable – across the different administrative domains that a cloud environment may span; (iii) be amenable to a series of automated correctness checks that increase assurance on their effectiveness. These requirements call for a novel framework for the definition of policies, one which accurately captures the knowledge that lurks behind policies and promotes a clear separation of concerns by disentangling the representation of policies from the actual code employed for enforcing them.

This paper proposes such a framework. More specifically, it proposes an *iterative multi-layered process* for the construction of an *ontological template* suitable for the *semantic representation* of policies in dynamic cloud environments. The proposed template is underpinned by a set of abstract relevant concepts, and their associations, that capture all those *knowledge artefacts* that are required for describing policies in a particular application domain. By undergoing a number of iterative refinement steps, this template is subsequently reified in order to arrive at readily enforceable concrete policies. For instance, regarding access control policies, the template would initially include such abstract concepts as 'subject', 'action', 'object', 'location of access', 'time of access', etc. These concepts would then be iteratively refined arriving, ultimately, at concrete access control policies. For instance, the 'location' concept could be initially refined by including sub-concepts for all geographical areas from which a sensitive data can be accessed and, subsequently, further concretised by including specific access locations as instances of these sub-concepts.

The proposed ontological template is expressed in terms of an extensible lightweight RDF vocabulary which lends itself to automated reasoning about the *correctness* of policies with respect to a set of relevant constraints on their actual content and structure. More specifically, it paves the way for the construction of a *higher-level ontology* which captures these *constraints* by drawing upon a richer formalism such as OWL 2 [5]. Both the policies and the constraints are therefore expressible in a uniform representation, namely as RDF graphs [6], hence facilitating the construction of a *policy validating* mechanism able to automatically assess the correctness of the policies.

Although the proposed framework is applicable to any kind of policy, in this paper we concentrate on the *security policies* devised as part of the PaaSword project [7]. This project sets out to offer a security-by-design solution – essentially a PaaS offering – assisting developers in defining *effective* security policies for dynamic cloud environments. PaaSword aims at *encryption policies*, that regulate the kind of cryptographic

978-1-5090-1445-3/16/$31.00 ©2016 IEEE (CloudCom'16)

protection that a sensitive data object should enjoy; (ii) *data fragmentation and distribution policies*, that articulate how data objects are fragmented and distributed over different physical servers in order to safeguard their privacy; (iii) *access control policies*, that specify the contextual circumstances under which access to sensitive data objects is granted or denied.

The rest of this paper is structured as follows. Section II presents related work. Section III presents our approach to the construction of the ontological template for the semantic representation of policies. Sections IV and V demonstrate how this template can be applied for modelling security policies in the PaaSword project. Finally, Section VI presents conclusions and future work

II. Related Work

Numerous works have endeavoured to address the deficiencies stemming from the absence of a proper separation of concerns between policy representation and policy enforcement [8–14]. In [8], a language for modelling security and management policies called PONDER is proposed; in a similar vein, the works in [9–11] embrace markup languages for formulating security (access control) policies. Nonetheless, such syntactic descriptions do not provide the means to capture the *knowledge* that resides in policies: they are simple data models that do not provide any semantic agreement beyond the boundaries of the organisations that adopted them. Any interoperability necessarily hinges upon the use of vocabularies that are shared among the parties involved in an interaction. This presents the following limitations: (i) it leads to ad-hoc reasoning about policy compliance – one that is inextricably tied to the specific vocabularies in which the rules of the reasoning are articulated; (ii) it restricts the portability and reusability of policies; (iii) it prohibits the identification of relations among policies (e.g. contradicting policies, pairwise subsuming policies, etc.); (iv) it restricts the ability to perform policy governance.

In order to overcome these limitations, semantically-rich approaches to policy specification have been proposed [12–14]. These generally employ *ontologies* in order to capture the *knowledge* that resides in policies. The rules that implement these policies can then be derived from this knowledge and can be articulated using any suitable syntactic description language. In [12], the authors present KAoS: a general-purpose policy management framework which exhibits a three-layered architecture comprising: i) a user interface layer; ii) a layer for managing and governing policies that uses OWL to express policy-related knowledge; iii) a policy monitoring and enforcement layer, which grounds OWL-expressed policies to a programmatic format that lends itself to policy-based monitoring and enforcement. In [13] the authors propose Rei – a language for specifying policies that draws upon OWL-Lite. Rei allows the articulation of a wide range of policies that discern those actions that *can* be performed, and those actions that *should* be performed on a resource. It therefore provides an abstraction that allows the expression of a desirable set of behaviours that are purportedly enforceable by different autonomous entities. In [14], POLICYTAB is proposed for allowing automated trust negotiation in Semantic Web environments. POLICYTAB advocates an ontology-based approach for specifying policies that regulate and drive a trust negotiation process aiming at granting controlled access to Web resources. These policies essentially determine the credentials that must be presented by an entity in order to perform an action on a resource owned by another entity.

The aforementioned approaches, whilst achieving a proper separation of concerns between policy specification and policy enforcement, rely on OWL's standard semantics which, due to the Open World Assumption [5], encumbers the definition of *constraints* on the actual content and structure of a policy [15]. This naturally hinders the construction of a *policy validator* that assesses the *correctness* of policies by checking their compliance against such constraints. In contrast, the reliance of our framework on RDF for the specification of policies absolves us from this limitation.

In [16], the development of a rule-based policy management system that can be deployed in the Web is described. The system combines the N3 language with a theorem prover designed for the Web in order to provide a mechanism for the exchange of access control rules, as well as of proofs that justify that these rules are indeed observed by an actor. Nevertheless, the proposed system does not provide any means for identifying inter-policy relations or performing any form of policy governance.

III. A Stepwise Approach to Generic Policy Representation

This work proposes an *ontological template* for the semantic representation of security policies. Such a template allows the generic formulation of any kind of policy whilst, at the same time, it lends itself to, and therefore paves the way for, a series of *correctness* checks that are performed automatically by a *policy validator* with reference to a higher-level ontology which captures constraints on the content and structure of the policies[1]. Below we outline a framework for the construction of such an ontological template. As depicted in Fig. 1, this framework comprises three phases or layers. At the first layer, the so-called *conceptualisation layer*, the identification and informal description of the concepts that are involved in the definition of a policy takes place. At the second layer, the so-called *formalisation layer*, an ontological template for the semantic respresentation of policies is derived by formalising the concepts identified in the first layer. At the third layer, the so-called *reification layer*, the ontological template of the second layer is instantiated to give rise to concrete policies. It is to be emphasised here that proceeding from one layer to the next is not necessarily an incremental process but may entail a number of *iterative passes* during which certain concepts that have already been formalised at the second layer are further specified, or *refined*, in terms of new, more fine-grained concepts; the identification of these finer-grained concepts takes place at the first layer before

[1] It is to be noted here that the purpose of this work is neither to present such a higher-level ontology, nor to present the policy validator.

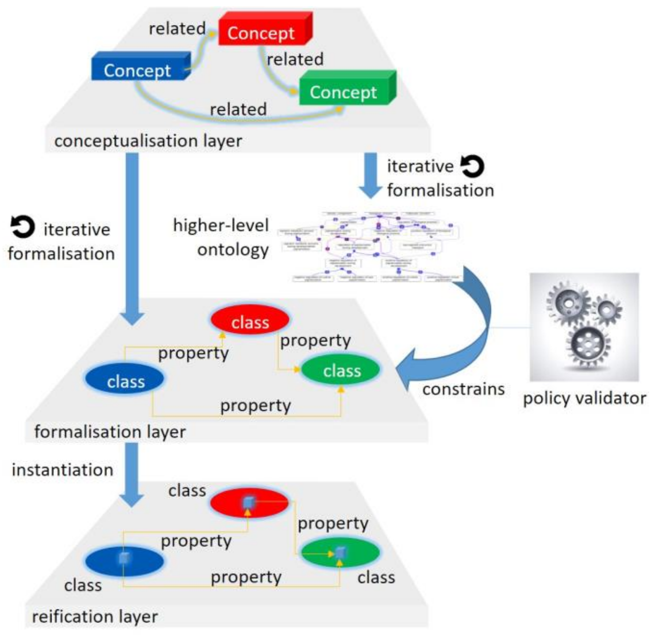


Fig. 1. Multi-layered approach to constructing an ontological template for modelling policies

prceeding anew to formalise them at the second layer. The three layers of our framework are further elaborated below.

A. Conceptualisation Layer

At the first layer, the *concepts* potentially involved in the definition of a policy, as well as their interrelations and associations with other concepts, are informally described. These concepts ultimately give rise to *attributes* whose values are constrained by the policy. Of course, the nature of these concepts, or attributes, depends on the underlying domain of discourse and the kind of policies that one is interested in defining. As an example, consider the case of context-aware access control policies. These policies are typically defined [17] in terms of such concepts as: the *physical location* from which an access request is performed, the *time* at which the request takes place, the *subject* that performs the request, the *kind of access* requested (e.g. read or write), the *object* (or resource) targeted by the request, etc.

It is to be noted that the concepts identified at the conceptualisation layer may be readily available from other ontologies that have already been devised in the underlying domain of discourse. For instance, in the realm of security, the USDL-SEC [18] ontology provides a number of concepts that may be readily used for modelling security policies.

Finally, it is to be emphasised that the conceptualisation layer has one more important role to play: by identifying the attributes whose values are constrained by policies, it essentially identifies the necessary, or optional, *ingredients* that any policy in the underlying domain of discourse encompasses, as well as the *value ranges* that these ingredients may attain. This paves the way for the definition of a set of *constraints* expressible in a *higher-level ontology* that determine the *correctness* of the policies[2]. For instance, going back to the access control example, a constraint may insist that any policy *must* specify *exactly one* value for the attributes *location*, *subject*, *object* and *kind of access*, and *may* specify *exactly one* value for the rest of the attributes; any policy not bearing these characteristics is not considered a correct policy. Similarly, a constraint may insist that the range of values that the location attribute may attain is drawn from a given set of specific locations – any policy specifying a location value outside this set is not considered a correct policy.

B. Formalisation Layer

At the second layer, the concepts and their associations identified at the conceptualisation layer are formalised in terms of an *ontological template*, one in which concepts are modelled as *classes* and interrelations as *properties*. In fact, as already discussed, such a formalisation may entail a certain degree of iteration and may be performed in a number of *passes*. Each pass comprises two distinct steps: (i) One which takes place at the conceptualisation layer and aims at identifying concepts and properties that further refine the classes and properties that already appear in the ontological template. (ii) One which takes place at the formalisation layer and aims at introducing a set of classes and properties that formalise the newly identified concepts and properties. This iterative process ceases when the concepts that already appear in the ontological template need not be, or cannot be, further refined at the chosen level of abstraction. Of course, if this level of abstraction changes and a more granular representation of concepts is favoured, new concepts that refine existing ones can be introduced anew. Going back to the access control example of Section III.A, a policy may initially be defined in terms of a set of top-level concepts including: the *physical location*, the *time* of day, the *subject*, the *kind of access* and the *object*. Each such concept is defined as a class in the ontological template. Nevertheless, some of these top-level concepts may prove overly abstract to be of practical use. A 2nd pass therefore ensues which identifies, at the conceptualisation layer, concepts that refine one or more of the existing concepts in the ontological template. For example, the physical location concept may be refined in terms of concepts that determine different ways of specifying locations (e.g. point coordinates, points of interest, geographical areas, etc.). These refining concepts are then formalised in terms of new classes and properties.

Finally, it is to be noted here that the classes and properties of the ontological template are also used by the higher-level ontology briefly discussed in Section III.A. In fact, these classes and properties form an adequate basis for the definition of domain-specific constraints concerning the allowable ingredients of a policy.

---

[2] The mere identification of the concepts that participate in a set of policies is not, by itself, sufficient for the construction of such a higher-level ontological framework. An entire body of constraints that restrict the different allowable forms of a policy must be articulated.

C. Reification Layer

At the third layer, a final reification step takes place whereby the classes and properties of the ontological policy template are *instantiated* in order to give rise to *concrete* policies. For instance, going back to the access control example, a concrete policy may be derived by instantiating the *subject* attribute with the individual 'Bob', the *location* attribute with the geographical coordinates that correspond to Bob's office, the *object* attribute with a particular data resource and the *kind of access* attribute with the value 'read/write'. Such a policy essentially grants to Bob read/write access to the particular data resource from his office.

In the following sections we shall demonstrate how the multi-layered framework presented above can be used for modelling a certain class of security policies, namely the PaaSword policies.

IV. Security Policies in the Cloud: An Abstract Framework

As stated in Section I, the PaaSword project discerns three kinds of security policy: *data encryption* policies, *data fragmentation and distribution* policies, and *access control* policies. The latter allow, or disallow, access to sensitive data objects on the basis of a set of *contextual attributes* pertaining to the entity requesting the access. Context awareness is considered of utmost importance for safeguarding access in dynamic and heterogeneous cloud environments [17]. *Attribute-based Access Control* (ABAC) policies, due to their inherent generality stemming from their reliance on the generic concept of an *attribute*, are deemed particularly suitable for infusing such contextual awareness [19]. They are thus adopted in PaaSword.

Following the multi-layered framework of Section III, we model the PaaSword security policies by initially identifying and informally describing a set of top-level concepts that are involved in the definition of these policies. These top-level concepts are subsequently formalised in terms of ontological classes and object properties.

A. PaaSword Policies: Conceptualisation Layer

The top-level concepts involved in the definition of the PaaSword policies are readily provided by USDL-SEC: Linked USDL's simple vocabulary for describing the security properties of an application [18]. This section provides a brief informal description of this vocabulary. Nevertheless, an outline of the reasons that led us to the decision to adopt the concepts offered by USDL-SEC is first in order.

1) Linked USDL. Linked USDL [18] is a remodelled version of USDL [20] which draws upon the results and experience gained with USDL, as well as with prior research efforts in the realm of Semantic Web Services and business ontologies [21]. It builds upon the principles of Linked Data in order to promote its use in a 'web of data'. In this respect, it expresses specifications in terms of an RDF vocabulary [22] that provides better support for the generic description of web and cloud services. Linked USDL comprises a Core schema, as well as a number of additional schemata, or profiles, addressing diverse business aspects of a cloud service. In this work we are interested in the Security profile (USDL-SEC) which provides an adequate basis for describing generically any security policy. The adoption of Linked USDL brings about a number of advantages [21]. Firstly, Linked USDL relies on existing widely-used RDF vocabularies, such as GoodRelations [23] the Simple Knowledge Organization System (SKOS) ontology [24], and FOAF [25]. In this respect, it promotes knowledge sharing whilst it increases the interoperability, and thus the reusability and generality, of our security policies. Secondly, by offering a number of different profiles, Linked USDL provides a holistic and generic solution able to adequately capture a wide range of business details. In addition, Linked USDL is designed to be easily extensible through linking to further existing, or new, ontologies. This is significant for our work for it facilitates seamless integration with the vocabularies devised in [26] for describing the concepts involved in the definition of the security policies (see Section V for more details).

2) Informal Description of the USDL-SEC Vocabulary. USDL-SEC identifies the five top-level concepts depicted in Fig. 2, namely Security Profile, Security Goal, Security Mechanism, Security Technology and Security Realisation Type. The Security Profile concept includes the different security profiles to which a cloud application may adhere. A security profile is tied to at least one security goal. This gives rise to the Security Goal concept that includes a set of sub-concepts each representing a distinct security goal – Fig. 2 depicts the list of security goals that are supported by USDL-SEC. A security goal is associated with the security mechanism through which it is implemented. This introduces the Security Mechanism concept that includes sub-concepts for representing particular security mechanism kinds – Fig. 2 depicts the list of security mechanism kinds that are provided by USDL-SEC. A security mechanism is associated with a particular security

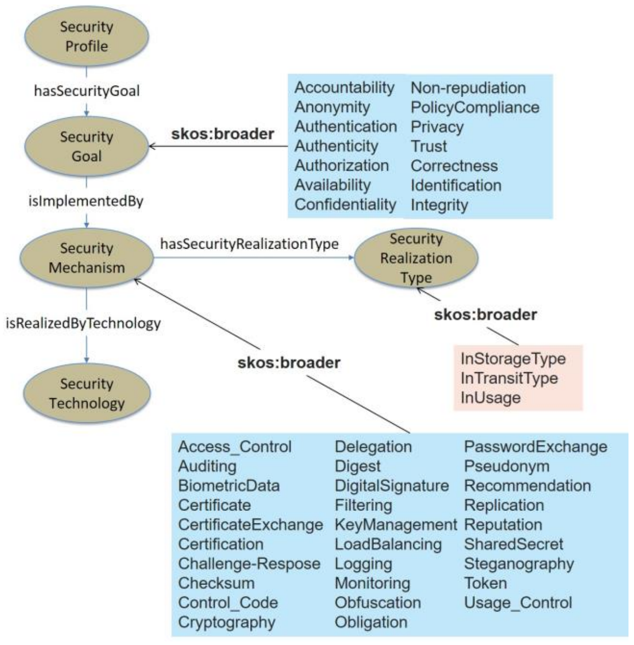


Fig. 2. USDL-SEC

technology that implements it, giving rise to the Security Technology concept. Moreover, a security mechanism is related to the particular layer of the ISO/OSI protocol stack at which it operates (for instance, the Application or Network layer); this brings about the Security Realization Type concept whose role is to determine this layer.

B. PaaSword Policies: Formalisation Layer

The concepts and their associations identified in the informal description of Section IV.A are formalised in terms of the USDL-SEC ontology depicted in Fig. 2. More specifically, each concept is formulated as a class of the ontology, and each concept association is formulated as an object property[3]. In particular, the following four object properties are discerned: `hasSecurityGoal` that links a security profile with its security goal(s); `isImplementedBy` that relates a security goal with the mechanism through which it is achieved; `isRealizedByTechnology` that links a security mechanism with the particular technology that realises it; `hasSecurityRealizationType` that determines the ISO/OSI layer at which the security mechanism operates. Furthermore, the SKOS broader property [24] is employed in order to express the fact that one concept forms a sub-concept of another. This framework of classes and properties forms an abstract ontological template that lays the foundations for constructing a model for the generic representation of the three kinds of security policy outlined at the beginning of Section IV. In Section V, we demonstrate how this framework is refined in order to derive an ontological template for modelling ABAC policies. The other two kinds of policy, namely data encryption and data fragmentation and distribution policies, are modelled analogously – the interested reader is referred to [27] for a relevant account.

V. ABAC Policies: An Abstract Framework

We model ABAC policies by refining the USDL-SEC concepts presented in Section IV. The refinement proceeds either by using existing classes that already appear in the USDL-SEC vocabulary (e.g. the *Security Mechanism* concept is refined in terms of its narrower *Access Control* concept – see Fig. 2), or by introducing new concepts, and their associations, that are subsequently formalised into ontological classes and properties. In the former case, the refinement is performed directly at the formalisation layer. In the latter case, the refinement requires an iteration between the conceptualisation and formalisation layers. In particular, the new concepts and their associations are identified and informally described at the conceptualisation layer and their subsequent formalisation in terms of classes and object properties takes place at the formalisation layer.

A. Refining USDL-SEC for ABAC Policies Using Existing Classes

ABAC policies are modelled under a particular security profile, namely the PaaSword Access Control (PAC) profile, which is represented as an instance of the class `SecurityProfile` (see Fig. 3)[4]. The security goal of this profile is *authorisation*. As depicted in Fig. 3, this is modelled by refining the `SecurityGoal` class in terms of the USDL-SEC `Authorization` class and defining an instance of this class, namely `AccessControlGoal`, to represent the particular authorisation goal. This goal is implemented by a security mechanism, in particular, an *access control* mechanism. This is captured by refining the `SecurityMechanism` class in terms of the USDL-SEC `AccessControl` class and defining an instance in this class, namely `AccessControlMechanism`, which represents the access control mechanism offered by the PaaSword framework. This mechanism operates at the *application layer* of the ISO/OSI protocol. This is captured by refining the `SecurityRealizationType` class in terms of the USDL-SEC `InUsageType` class[5] and introducing an instance of this class, namely `AccessControlType`.

B. Refining USDL-SEC for ABAC Policies by Introducing New Concepts

The PaaSword access control mechanism is implemented in terms of a security technology that is based on a particular ABAC model, one which takes into account the context from within which an access request is made. In order to model such a security technology, a number of new concepts, and their associations, need to be introduced and described at the conceptualisation layer. These are the *PaaSword ABAC* concept, which represents the particular ABAC technology utilised, and the *ABAC Policy Set* concept, which bundles together the ABAC policies on which the *PaaSword ABAC* concept relies. These two concepts are formalised by introducing the classes `PaaSwordABAC` and `ABACPolicySet` respectively (see Fig. 3). These classes are interrelated through the object property `hasABACPolicySet`. The instance `AccessControlTechnology` represents the particular security technology that the PaaSword project utilises.

C. Further Refining USDL-SEC for ABAC Policies

The `ABACPolicySet` concept introduced above encompasses the ABAC policies on which PaaSword relies. This concept must be further refined in order to derive an ontological model of the actual ABAC policies that it comprises. Such a refinement entails the introduction of a set of new concepts and their associations. It thus entails a new iteration between the conceptualisation and formalisation layers. In particular, the new concepts are identified and informally described at the conceptualisation layer and are subsequently formalised to make their way into the ontological template of Fig. 3.

[3] All USDL-SEC classes and properties are prefixed with the `usdl-sec` namespace. This namespace is omitted here to reduce notational clutter.

[4] All classes, properties and instances introduced as part of the formalisation and reification of the PaaSword Access Control profile are drawn from the `pac` namespace. To reduce notational clutter, this namespace is omitted from all textual descriptions (it appears though in the figures).

[5] This class represents one of the classes offered by USDL-SEC.

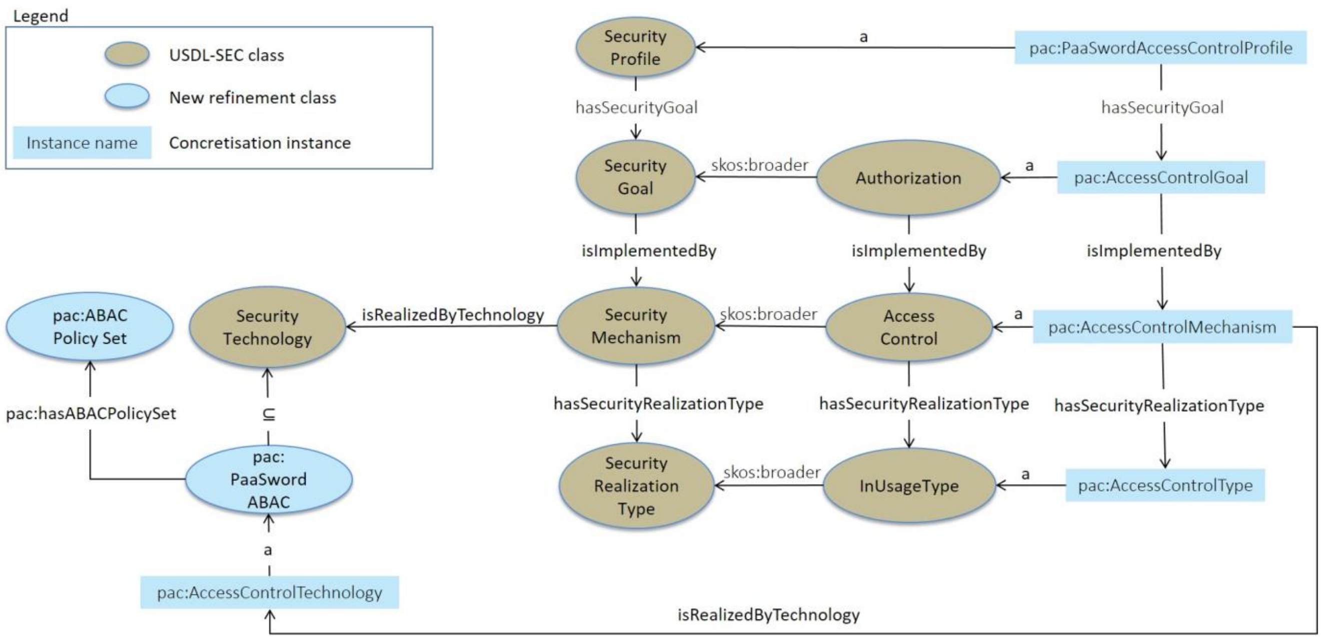


Fig. 3. USDL-SEC refinement

The conceptualisation and formalisation of these new concepts is further elaborated below.

1) Conceptualisation Layer. Following an approach inspired by the XACML standard [8], ABAC policy sets are refined by associating them with the actual policies that they comprise which are, in turn, associated with sets of rules. In this respect, the concepts *ABAC Rule* and *ABAC Policy* are introduced. The ABAC Rule concept is the basic building block of an ABAC policy. It represents the template depicted in Table I. This template defines a generic structure to which all PaaSword ABAC rules adhere. It comprises the attributes *Actor*, *Context Expression*, *Auth*, *Action* and *Controlled Object* which are represented in terms of a set of corresponding concepts. Thus, the concept *Actor* identifies the subject requesting to carry out an operation on a data object; the concept *Context Expression* represents an expression that determines the contextual conditions that must be satisfied in order to allow, or disallow, an operation on a data object; the concept *Auth* specifies the kind of authorisation (i.e. 'permit' or 'deny') that is granted; the concept *Action* articulates the requested operation; the concept *Controlled Object* identifies the sensitive object.

TABLE I. RULE TEMPLATE

[*Actor*] with [*Context Expression*] has [*Auth*] for [*Action*] on [*Controlled Object*]

2) Formalisation Layer. The concepts ABAC Rule and ABAC policy are formalised in terms of the classes `ABACRule` and `ABACPolicy` respectively – see Fig. 4. These classes are interrelated through the object property `hasABACRule` which associates an ABAC policy with its constituent rules. In addition, the `ABACPolicy` class is associated with the policy set to which it belongs through the property `belongsToABACPolicySet`. The rest of the concepts informally presented above are formalised as follows. The Controlled Object concept is formalised in terms of the class `Object`. This class is associated with an ABAC rule through the object property `hasControlledObject`. It is part of the Context Model [17] [26] – a vocabulary of relevant classes and properties that has been devised in the PaaSword project for describing the various contextual attributes that may appear in an ABAC rule[6]. For reasons of space, the Context Model (CM) is not further elaborated here. The interested reader is referred to [26] for more details. The concept Auth is formalised in terms of the class `Authorisation`. This class is associated with an ABAC rule through the object property `hasAuthorisation`. It invariably comprises the two instances depicted in Fig. 4. The concept Action is formalised in terms of the class `Permission`. This class belongs to the CM and is associated with an ABAC rule through the object property `hasAction`. The concept Actor is formalised in terms of the class `Subject`. This class belongs to the CM and is associated with an ABAC rule through the object property `hasActor`. Finally, the concept Context Expression is formalised in terms of the CM class `ContextExpression` which is associated with an ABAC rule through the object property `hasContextExpression`. This class is further elaborated in Section V.D below.

[6] The `pcm` namespace, as well as the `ppm` and `pcpm` namespaces (see Fig. 4 and Fig. 5), are defined as part of the Context Model [17], [26]. To reduce notational clutter, these namespaces are again omitted from textual descriptions.

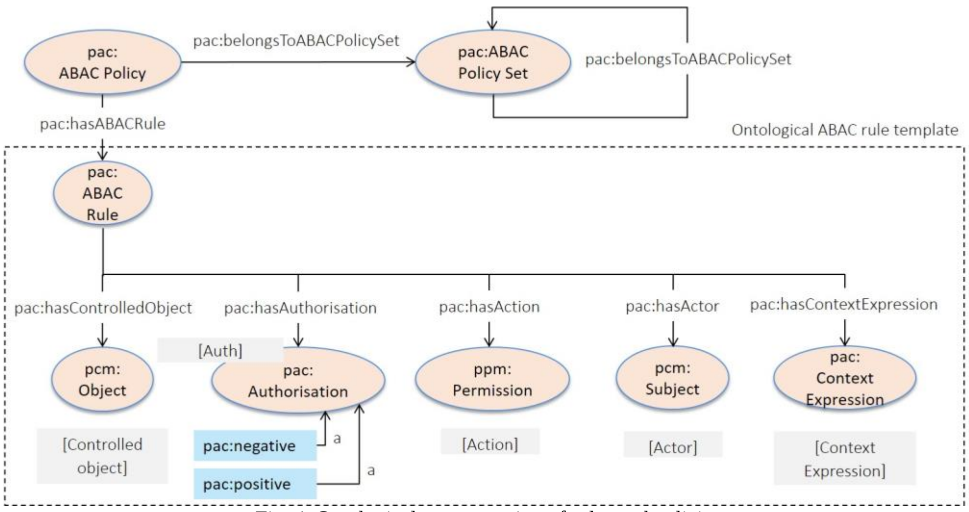


Fig. 4. Ontological representation of rules and policies

D. Context Expressions for ABAC Policies

The class `ContextExpression` specifies the contextual conditions that must be satisfied in order to allow, or disallow, an operation on a sensitive object. Any further specification of this concept requires the introduction of new concepts and their associations. It thus entails a new iteration between the conceptualisation and formalisation layers as outlined below.

1) Conceptualisation Layer. The context expression is founded on the `ContextPattern` and `SecurityContextElement` concepts. The former identifies any historical access patterns that need to be considered in order to permit, or deny, an access request. Such patterns are defined in terms of a number of contextual attributes such as the 'physical location' pattern, the 'date and time' pattern, the 'type of device of access' pattern, etc. The latter identifies the contextual attributes, as well as their allowable values and value ranges, that are taken into account in order to decide whether to permit, or deny, an access request.

2) Formalisation Layer. The two concepts informally outlined above are formalised in terms of the CM classes `ContextPattern` and `SecurityContextElement` respectively (see Fig. 5). These classes are associated with the class `ContextExpression` via the object properties `hasPatternParameter` and `hasParameter` respectively (Fig. 5). A context expression may be defined recursively by logically combining one or more other context expressions. This is captured through the `hasParameter` and `hasPatternParameter` properties that relate the `ContextExpression` class with itself. It is also captured by the four subclasses of `ContextExpression` depicted in Fig. 5 which indicate the kinds of logical connectivity used. This is better explained through an example. Suppose a context expression (call it `expr`) that allows access to a hypothetical sensitive object only during workdays and not from the location "IMU premises". As depicted in Fig. 6, `expr` is an instance of the class `ANDContextExpression` and thus it is the logical conjunction of two parameters: the `IsWorkday` parameter and a nested context expression. The `IsWorkday` parameter is an instance of the CM class `DateTimeInterval` and represents a time interval restricted through the CM properties `hasBeginning` and `hasEnd`. The nested context expression is an instance of the class `NOTContextExpression` and thus logically negates the `InIMUPremises` parameter. This parameter is an instance of the CM class `AbstractLocation` and is restricted through the CM properties `hasLatitude`, `hasLongitude` and `hasRadius` to represent a particular geographical area.

VI. Conclusions

This paper has proposed a multi-layered iterative framework for the construction of an ontological template for the semantic representation of security policies in dynamic and heterogeneous cloud environments. We argue that this framework facilitates developers in expressing effective security policies which give rise to appropriate security controls that safeguard sensitive data in the cloud. One of the virtues of the proposed ontological template is that it is expressed in a generic, interoperable and extensible RDF vocabulary that lends itself to, and therefore paves the way for, a series of correctness checks that are performed automatically by a policy validator. These checks aim at assessing the validity of a policy with respect to a higher-level ontology that

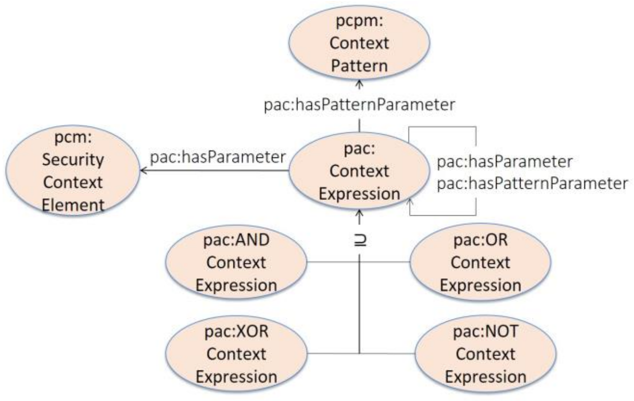
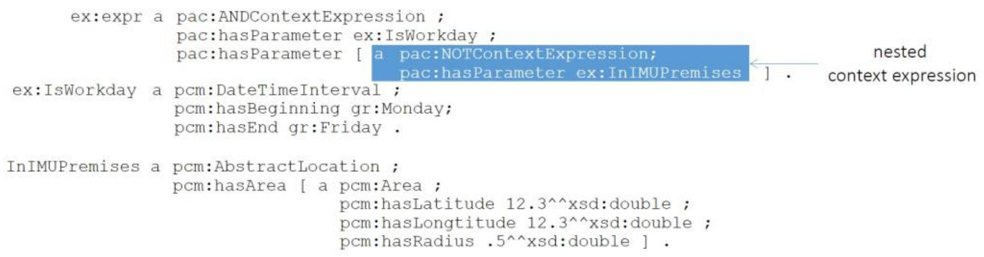

pcpm: Context Pattern
pac:hasPatternParameter
pcm: Security Context Element
pac:hasParameter
pac: Context Expression
pac:hasParameter
pac:hasPatternParameter
⊇
pac:AND Context Expression
pac:OR Context Expression
pac:XOR Context Expression
pac:NOT Context Expression

Fig. 5. Ontological representation of a context expression



```
    ex:expr a pac:ANDContextExpression ;
              pac:hasParameter ex:IsWorkday ;
              pac:hasParameter [ a  pac:NOTContextExpression;
                                    pac:hasParameter ex:InIMUPremises  ] .
ex:IsWorkday  a pcm:DateTimeInterval ;
              pcm:hasBeginning gr:Monday;
              pcm:hasEnd gr:Friday .

InIMUPremises a pcm:AbstractLocation ;
              pcm:hasArea [ a pcm:Area ;
                                pcm:hasLatitude 12.3^^xsd:double ;
                                pcm:hasLongtitude 12.3^^xsd:double ;
                                pcm:hasRadius .5^^xsd:double ] .
```

Fig. 6. Context expression modelling example

captures all those ingredients that a policy may, or may not, comprise. These correctness checks are clearly of utmost importance for they increase assurance on the effectiveness of the policies.

As part of future work, we intend to construct the higher-level ontology. This ontology essentially constitutes a schema which imposes a number of constraints on the RDF statements that may, or may not, be encountered in a policy that has been formulated according to the multi-layered framework of Section III. These constraints are expressed using the Integrity Constraints (IC) semantics for the OWL 2 Web Ontology Language proposed in [15]. A policy validator that parses the higher-level schema, as well as an input policy, and determines whether the latter complies with the former may then be constructed. Moreover, as part of future work, we intend to construct an editor through which a user will be able to 'prime' the higher-level ontology with appropriate constraints for a particular domain of application.

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644814. The authors would like to thank Dr Yiannis Verginadis and Dr Ioannis Patiniotakis for their contribution to the formalisation of context expressions, as well as the anonymous reviewers for their valuable comments.

REFERENCES

[1] Cloud Computing Reference Architecture. Technical report, NIST (2011)

[2] "What's Hindering the Adoption of Cloud Computing in Europe?," 15 September 2015. [Online]. Available: https://blog.cloudsecurityalliance.org/2015/09/15/whats-hindering-the-adoption-of-cloud-computing-in-europe/

[3] CloudPassage, "Cloud Security Spotlight Report," LinkedIn, 2015

[4] Veloudis, S., Paraskakis, I., Petsos, C.: Cloud Service Brokerage: Strengthening Service Resilience in Cloud-Based Virtual Enterprises. In Camarinha-Matos et al. (eds.) PRO-VE 2015. LNCS, vol 463, pp. 122--135, Springer, Heidelberg (2015)

[5] OWL 2 Web Ontology Language Primer (2nd Edition), https://www.w3.org/TR/owl2-primer/

[6] RDF 1.1 XML Syntax, http://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/

[7] PaaSword project, http://www.paasword.eu/

[8] Damianou, N., Dulay, N., Lupu, E., Sloman, M.: The Ponder Policy Specification Language. In Sloman, M., Lobo, J., Lupu, E. (eds.) In *Proceedings of the International Workshop on Policies for Distributed Systems and Networks* (POLICY '01), pp. 18-38, Springer-Verlag, London (2000)

[9] eXtensible Access Control Markup Language (XACML) Version 3.0. 22 January 2013. OASIS Standard. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html

[10] Security Assertions Markup Language (SAML) Version 2.0. Technical Overview 25 March 2008. OASIS Standard. https://www.oasis-open.org/committees/download.php/27819/sstc-saml-tech-overview-2.0-cd-02.pdf (2008)

[11] WS-Trust 1.3. 19 March 2007. OASIS Standard. http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.doc (2007)

[12] Uszok, A., Bradshaw, J., Jeffers, R., Johnson, M., Tate, A., Dalton, J., and Aitken, S.: KAoS Policy Management for Semantic Web Services. IEEE Intel. Sys. 19, 4, 32--41 (2004)

[13] Kagal, L., Finin, T., Joshi, A.: A Policy Language for a Pervasive Computing Environment. In 4th IEEE Int. Workshop on Policies for Distributed Systems and Networks (POLICY '03), pp. 63--74, IEEE Computer Society, Washington, DC (2003)

[14] Nejdl, W., Olmedilla, D., Winslett, M, Zhang. C.C.: Ontology-Based policy specification and management. In Gómez-Pérez, A. and Euzenat, J. (eds.) ESWC'05, pp. 290-302, Springer-Verlag, Berlin, Heidelberg (2005)

[15] Tao, J., Sirin, E., Bao, J. and McGuinness, D. L.: Integrity Constraints in OWL, *In Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI-10),* Atlanta, Georgia, USA, July 11-15 (2010)

[16] Weitzner, D. J., Hendler, J., Berners-Lee, T. and Connolly, D.: Creating a Policy-Aware Web. Web and Information Security, pp. 1–31. DOI: 10.4018/978-1-59140-588-7.ch001 (2006)

[17] Veloudis, S., Verginadis, Y., Patiniotakis, I., Paraskakis, I., Mentzas, G.: Context-aware Security Models for PaaS-enabled Access Control. CLOSER Conference (2016)

[18] Linked USDL, http://www.linked-usdl.org/

[19] Hu, V. C., Ferraiolo, D., Kuhn, R., Schnitzer, A., Sandlin, K., Miller R., and Scarfone K.: Guide to Attribute Based Access Control (ABAC) Definition and Considerations. NIST (2014)

[20] Barros, A. and Oberle, D.: Handbook of Service Description: USDL and its Methods, Springer (2012)

[21] Cardoso, J., Pedrinaci, C., Leidig, T., Rupino P. and Leenheer, P.: Foundations of Open Semantic Service Networks. *International Journal of Service Science, Management, Engineering, and Technology,* vol. 4, no. 2, 1-16 (2013)

[22] Cardoso, J., Pedrinaci, C., Leidig, T.: Linked USDL: a Vocabulary for Web-scale Service Trading. In *11th Extended Semantic Web Conference (ESWC)* (2014)

[23] GoodRelations: The Professional Web Vocabulary for E-Commerce. http://www.heppnetz.de/projects/goodrelations/

[24] SKOS Simple Knowledge Organization System. http://www.w3.org/2004/02/skos/

[25] The FOAF Project. http://www.foaf-project.org/

[26] PaaSword Deliverable 2.1. https://www.paasword.eu/deliverables/

[27] PaaSword Deliverable 2.2. https://www.paasword.eu/deliverables/